Assignment 2 Neighborhood Processing & Filters

Daniel van de Pavert(11045418), Shuai Wang(13128051), Damiaan Reijnears (10804137)

September 2020

1 Introduction

In this assignment the focus lies on convolutions, local structure and various applications of filters. First we take a closer look at convolutions and the relations convolutions and correlations have. Later we use convolutions to apply various filters and further look into the effects of these filters. The filters that are the main focus are the Gaussian filter and the Gabor filter. The application of the techniques expanded upon in this assignment is done in denoising, edge detection and the separation of foreground and background.

2 Neighborhood Processing

Convolutions

F rom an analytical point of view, convolution and correlation are equivalent operators with mirrored kernels. Algebraically, the formulas differ only with regard to the '*plus*-' and '*minus*-sign.' However, the major practical difference between the correlation and convolution operators is the fact that the convolution operator satisfies the *associative* algebraic property, while correlation does not $(f * (g * h) = (f * g) * h$, while $f \star (g \star h) \neq (f \star g) \star h)$. This is a very useful property as it allows for *convolving of convolution kernels itself*, which significantly decreases the amount of computation which is needed to perform two consecutive convolutions on an image. An example of this is the operation of *smoothing* an image before 'taking the derivative' in a certain direction of the image (for the latter mentioned operation, we define the subtraction of a pixel intensity value with a neighbouring pixel value, or a similar operation further elaborating on this idea) – instead of looping through all the pixel intensity values of the image twice (once to perform smoothing, once to 'take the derivative'), we can take the derivative of the smoothing convolution kernel (or vice-versa, as the convolution operator also satisfies the *commutative* property) and loop through the image only once.

C onvolution and correlation can actually be proven to be equivalent if we allow the respective kernels to be *opposites* (the altered correlation kernel will become the *negated* convolution kernel, and vice-versa). From the formulas of convolution and correlation, as illustrated in equation 1 and 2,

$$Conv(x,y) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} I(x-k, y-l) \cdot \boldsymbol{h}(k,l) \tag{1}$$

$$Corr(x,y) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} I(x+k, y+l) \cdot \boldsymbol{h}(k,l) \tag{2}$$

If we set $u = -k$ and $v = -l$, then, derived from equation 1 which is a convolution, we can obtain equation 3, which is correlation and thus equivalent to the convolution (with an altered mask h).

$$\sum_{u=-1}^{1} \sum_{v=-1}^{1} I(x+u, y+v) \cdot \boldsymbol{h}(-u,-v) \tag{3}$$

Proof of computational advantage

This subsection effectively answers question 2 of the lab exercise, which was originally part of chapter 3 - 'Low-Level filters.'

If we assume the separable convolutional operation of the *Gaussian blur* (of which the proof for its separability is beyond the scope of this question), $G = G_x * G_y$, with a scale σ, then the computational advantage of a separable convolution kernel (which generally means the *separation* of a 2D kernel of size $m \times m$ into two 1D kernels, each the size of $1 \times m$ and $m \times 1$) can be shown analytically by noting that—if we would think in terms of matrices—G is equal to the matrix product of G_x and G_y, which consists of σ^2 elements, while the sum of both 1D Gaussian filters results in 2σ elements. Since we can interchangeably use the terms of 'kernel' and 'matrix' for the sake of showing the computational advantage, we can further note that the size of the kernel determines the cost of computation. More formally, we can define the computational complexity of convolution with a 2D Gaussian filter as $O(\sigma^2)$, as opposed to $O(2\sigma)$ for twice convolving with 1D Gaussian filters.

3 Low-Level filters

3.1 Gaussian filters

Use of second-order derivatives

The second order derivative is used for **edge detection**. As the derivative of a function measures the function's 'sensitivity to change' of its input variable, expressed by a ratio with respect to its output value, the derivative is formally defined using a limit for the change of its input argument approaching zero. Furthermore, as an 'image' is defined as a finite sequence of intensity values of the pixels it consists of, a function from its pixel coordinates to their corresponding intensity values has a discrete domain (assuming no interpolation techniques[1] are implemented whatsoever). The 'derivative' of an image is taken over its discrete pixel values, which is equivalent to subtracting a pixel value from its successor pixel value in the x or y direction (as, for now, we only assume two-dimensional images). The derivative of an image then measures 'change in the successor direction,' which generally correspond to the *edges* of objects in an image (as *background* 'suddenly' flows into *foreground* or vice-versa). The second derivative (a derivative, following the same procedure, of the newly obtained derived image) then measures change in the pixel values which already correspond to changes in pixel intensity of the original image. It follows that, as, in the first derived function over the pixels of an image, an edge is represented by the first derivative as an *extrema* (a local *maximum* or *minimum*) the second derivative represents this same edge at a *zero crossing* (as this is the point where a function, after a local maximum or minimum has been encountered, either continues in (respectively) descending or ascending direction).

However, second derivatives are more prone to noise as noisy data tends to get magnified when taking derivatives of the second order[2], which implicates that additional *smoothing* (with a larger kernel) should be performed in order to minimize this effect. Furthermore, the latter proposed solution has its own disadvantages – the higher the degree of smoothing (the larger the *scale*, or kernel), the bigger the loss of detail in the image.

3.2 Gabor filters

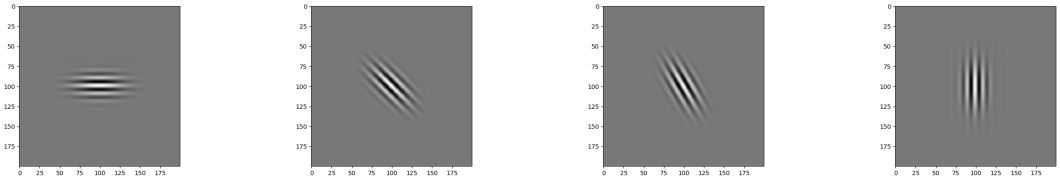
The major difference between Gabor filters and other methods for the purpose of edge detection is the fact that Gabor filters do *not* rely on derivatives of pixel intensity values, but rather on the geometric properties of *sinusoidal functions*[3], which makes it more suitable for extracting edges as it is a more *robust* 'family' of filters – edges come in different shapes, scales, and 'blurs,' which can be captured less easily with derivatives as they are always 'directional,' while cosine waves can be oriented and scaled towards all possible types of edges. The Gabor filter essentially constructs

[1] In many applications it is actually needed to assume a continuous function over the domain of pixel values, for which intensity values—which were not actually measured by a sensor—are 'estimated.'

[2] StackExchange, *Are 2nd Order Edge Detectors More Susceptible to Noise?*, `https://dsp.stackexchange.com/a/30386/52340`, accessed on September 18th, 2020

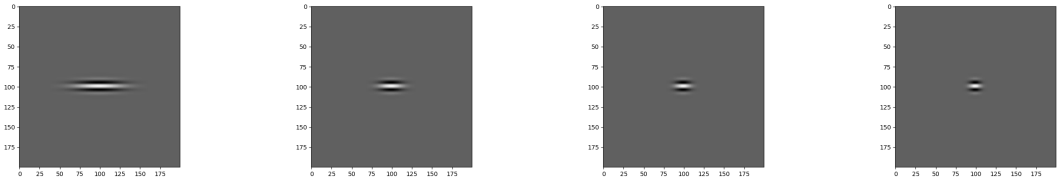
[3] The Gabor filter the application of a *Gabor transform*, which is, in essence, a *Fourier transform* multiplied by a Gaussian distribution, so that it focuses on local tendencies. The Fourier transform itself is a method for *decomposing* a (sinusoidal) function into separate functions. The fundamental difference between Fourier transforms and Gabor transforms is the fact that the pixel coordinates—obviously a necessity when detecting edges—are preserved as the Gaussian effectively creates a 'window function.'

cosine waves based on the pixel intensity values and pixel coordinates of an image, and then selects 'waves of interest' (intuitively, the 'inner workings' of Gabor filters can be related to the way how audio signals can be separated, from which certain 'sub signals' can then be extracted). This 'wave extraction procedure,' which essentially extracts image features, in the capacity of *texture features* (since we're using sinusoid wavelets, which implies repeated occurrences of certain features), can and need to be 'tweaked' towards the 'waves of interest.' This means that we extract one 'type of feature' at every loop, and implicates that, if we want to extract multiple features from a single image, we need to perform multiple Gabor filters on the image.



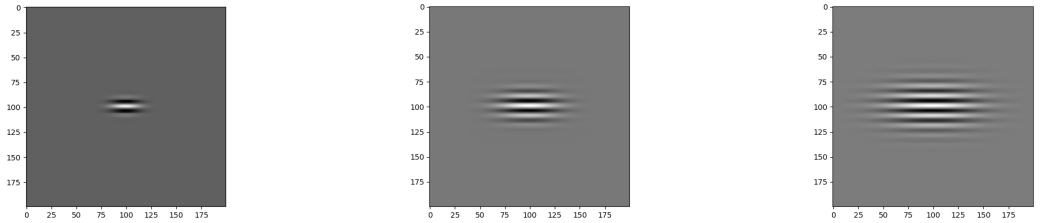
(a) $\theta = 0$ (b) $\theta = \frac{\pi}{4}$ (c) $\theta = \frac{\pi}{3}$ (d) $\theta = \frac{\pi}{2}$

Figure 1: Effects of varying the orientation parameter, θ, with $\lambda = 10, \psi = 0, \sigma = 10, \gamma = 0.5$



(a) $gammaa = \frac{1}{4}$ (b) $\gamma = \frac{1}{2}$ (c) $\gamma = \frac{3}{4}$ (d) $\gamma = 1$

Figure 2: Effects of varying the orientation parameter, γ, with $\lambda = 10, \psi = 0, \sigma = 5, \theta = 0$



(a) $\sigma = 5$ (b) $\sigma = 10$ (c) $\sigma = 15$

Figure 3: Effects of varying the orientation parameter, σ, with $\lambda = 10, \psi = 0, \theta = 0, \gamma = 0.5$

If we consider the filter *itself* as an image, we can easily indicate the role of the Gabor filter's parameters:

Wavelength

The Gabor filter is, as explained, a product of a cosine function with a Gaussian distribution. The Wavelength parameter, usually denoted by λ, controls the wavelength of the waves produced by this cosine function. The larger the wavelength, the larger the 'window.'

Orientation

The orientation parameter, θ, controls the orientation of the produced 'strips' in the filter itself. It is worth to note that the pixel coordinates are multiplied by the cosine and sine function of θ, and influence both the Gaussian- and cosine part of the Gabor function. The effect of this parameter is illustrated in figure 1.

Phase offset

The phase offset, ψ, is simply the offset of the cosine function, and is generaly set at 90 degrees. However, differences in this parameter influence the 'type of features' (e.g. lines or edges) that are extracted[4].

Aspect ratio

The aspect ratio parameter, denoted as γ, is a parameter influencing the Gaussian-part of the equation, and thus influences the 'window of focus' of the Gabor filter – the aspect ratio models the shape of the Gaussian function (usually an ellipse). The influence of this parameter on the model is illustrated in figure 2.

Bandwidth

The bandwidth parameter, σ, can be roughly interpreted (intuitively) as the *variance* of the Gaussian function. It is no surprise that this parameter controls the size of the Gaussian 'envelope' (the 'window of focus'). In order to explain this more clearly: the Gabor filter models an image's pixel values as *sine waves* which are multiplied by a Gaussian function, as sine waves are inherently infinite, and the Gaussian distribution allows us to focus on only a certain part of the sine wave (which is composed of multiple waves). The effect of varying the bandwidth is shown in figure 3.

4 Applications in image processing

4.1 Noise in digital images

4.2 Image denoising

Q6.1: Peak Signal to Noise Ratio (PSNR) are used to compare the squared error between the original image and the reconstructed image. There is an inverse relationship between PSNR and MSE. So a higher PSNR value indicates the higher quality of the image (better).

Q6.2: PSNR between image1-saltpepper.jpg and image1.jpg is 16.108.

Q6.3: PSNR between image1-gaussian.jpg and image1.jpg is 20.584.

Q7.1: Figure 3 and Figure 4 show results of denoising image saltpepper.jpg and image1 gaussian.jpg by different method.

[4]StackOverflow, *In open CV Why is the default Gabor phase offset 90 degrees?*, `https://stackoverflow.com/questions/25822667/in-open-cv-why-is-the-default-gabor-phase-offset-90-degrees`, accessed on September 21st, 2020

Figure 4: Saltpepper image with Box and Median filter

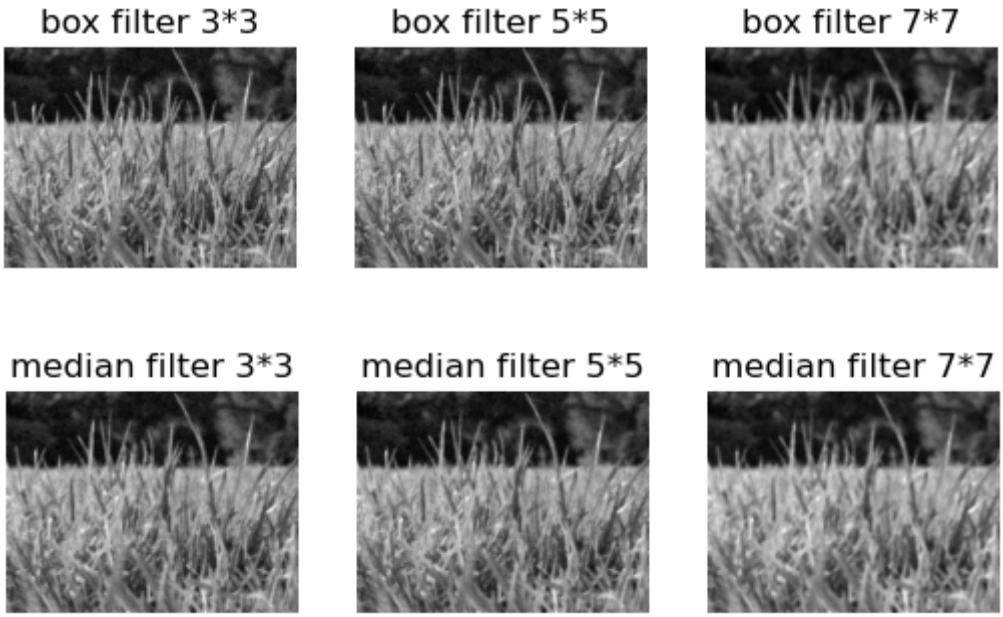


Figure 5: Gaussian image with Box and Median filter

Q7.2: From Table 1 we can see that the PSNR becomes smaller when kernel size grow larger.

	Saltpepper image	Gaussian image
Box 3*3	28.965	28.431
Box 5*5	28.775	28.317
Box 7*7	28.637	28.253
Median 3*3	30.147	29.012
Median 5*5	29.523	28.540
Median 7*7	29.203	28.381

Table 1: PSNR with different filter ways

Q7.3: For saltpepper images, median filter are better since their PSNR numbers are greater. For Gaussian images, Box and Median's effect are similar and Median works a little better.

Q7.4: In figure 6, We choose Gaussian kernel of 3*3 as size 3*3 performs best for former box and median filter. In order to conclude this we compared results from different Sigma of Gaussian filters.

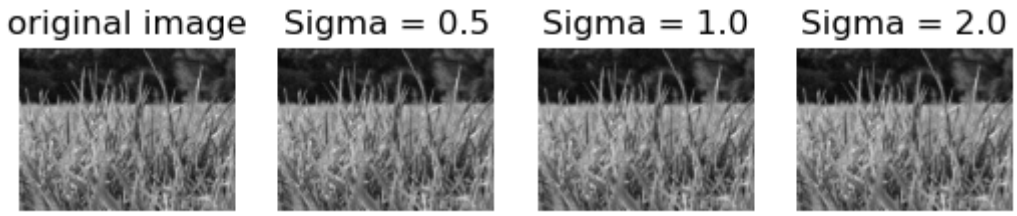


Figure 6: Gaussian image with different Sigma in 3*3 kernel

Filter	Sigma	PSNR result
Gaussian 3*3	0.5	35.150
Gaussian 3*3	1.0	28.568
Gaussian 3*3	2.0	28.461

Table 2: PSNR results with different Sigma value

Q7.5: From Table 2, we notice PSNR becomes smaller as Sigma become larger.

Q7.6: Box filter simply takes the average of all the pixels under the kernel area and replaces the central element. Besides, the Median Blurring takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image. Finally, for Gaussian, instead of a box filter, a Gaussian kernel is used. The width and height of the kernel should be specified, which means positive and odd. We also should specify the standard deviation in the X and Y directions(sigma). If two filtering methods give a PSNR in the same ballpark, we can still notice a qualitative difference. Because PSNR just show difference between original image and the reconstructed image. When every pixel in the image has the potential to change, the whole image can change a lot.

4.3 Edge detection

Q8: In Figure 7, as you can see Gx is just difference between pixels intensity of points to east and west of center point. Similarly Gy is difference in pixel intensity of points to south and north of central pixel. The magnitude of the gradient tells us how quickly the image is changing, while the direction of the gradient tells us the direction in which the image is changing most rapidly. This information is helpful for detecting edges in latter process.

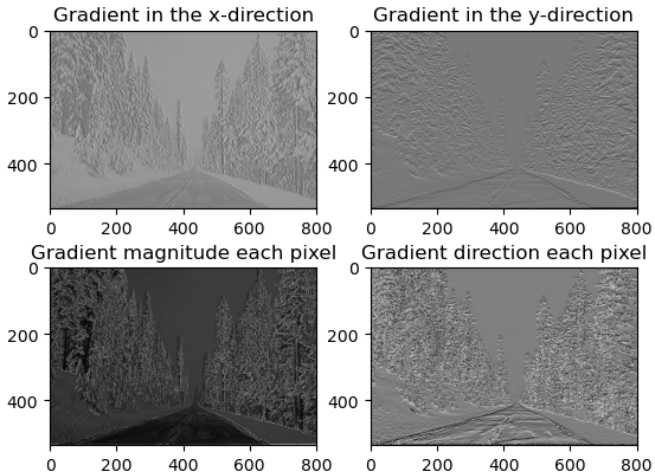


Figure 7: Gradient in each direction and pixel

Q9.1: Visualization Results of three methods are shown in Figure 8.

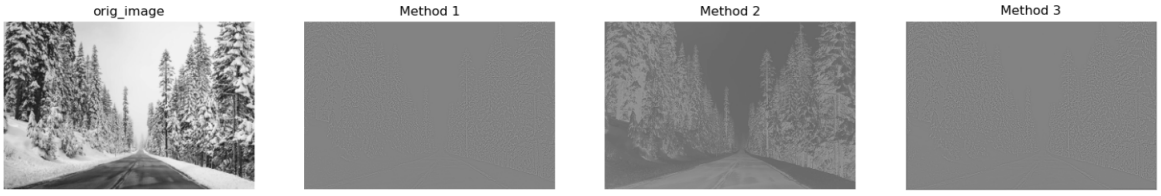


Figure 8: Gradient in each direction and pixel

Q9.2 - Q9.4: Methods 1 and 2 involve computing the *Laplacian* (the 'divergence of the *gradient*' at every pixel x, y in the image) after having *smoothed* the image with a Gaussian (as to reduce noise, since the Laplacian is extremely prone to noise as it involves second partial derivatives, as explained in section 3.1 of this document) – the *Laplacian of Gaussian*-method (LoG). Method 1, however, is very inefficient compared to method 2, while both methods are equivalent. Its computational efficiencies and equivalence is proven in section 2 of this lab report. In many popular real-world applications, such as in (Lowe, 2004), the *Difference of Gaussians*-method (DoG) is used. DoGs involve computing the difference (subtraction if intensity values of pixels) of two Gaussian kernels, and is shown to be roughly equivalent to LoG. For the purpose of edge detection in images however, the results of LoG and DoG will be *exactly equivalent* for the purpose of *locating a boundary*, as the only difference in the DoG's approximation the the LoG is a *multiplicative constant*, from which it is clear that extreme points in an image (which we use for finding boundaries in the original image) will remain 'extreme' (as simply *all* pixels will be multiplied by the same constant) (Derpanis, 2006).

As both the LoG and DoG involve a single kernel with which the image has to be convolved, there is no computational advantage for using the DoG, compared to the LoG, for the convolution itself. Moreover, both the LoG and DoG are *not* separable (although a Gaussian kernel is separable, a DoG is not separable as it involves subtraction of two kernels)[5]. However, if applying the filter at *multiple scales*, the DoG is significantly more efficient as it decreases *computational redundancy*[6] In order to adequately accentuate 'features' of an image occurring on different scale levels, while preserving *their own characteristic features*, a *scale space* can be used – this effectively means that an image is analyzed with respect to different smoothing scales. The calculation of the LoG kernel is computationally more expensive (as it involves second partial derivatives, since we're dealing with the *Laplacian*).

As the DoG involves subtraction of two images separately convolved with a Gaussian filter of different scales, each set of scales σ_1 and σ_2 will yield different results, and will thus not equivalently approximate the LoG. The smaller the ratio between the scale values, the better the approximation the log – which is an adequate answer to the question of which ratio between σ_1 and σ_2 is best. But, as different factors often come into play, the best approximation is not necessarily the best *applicable* approximation. There seems to be general consensus among the scientific community regarding this value, however: the best ratio between σ_1 and σ_2 is **1.6** [7] (Marr and Hildreth, 1980, p. 187-217).

Q9.5: Improvements

There are different approaches if thinking about improvements of the LoG/DoG methods. If 'keeping close to the currently proposed methods,' one could think of certain *pre-processing* methods (such as colour-based thresholds) to reduce noise and increase precision. A totally different direction is to implement approaches which are not based on taking derivatives (on which, analytically, both the LoG and DoG methods are based), but rather on wavelets (such as the *Daubechies wavelet*).

[5] A small note has to be made on this, since the DoG can technically be separated into four one-dimensional Gaussians, just like the LoG.

[6] In a *Gaussian scale space*, an image is divided into multiple *octaves* of different scales of the original image (each octave is a down scaled version of the previous octave) which are successively convolved with a Gaussian filter, which can then later be subtracted from each other to obtain the DoG).

[7] A general overview focusing on this ratio is given by (Winnemöller et al., 2012) in a standalone appendix: `https://users.cs.northwestern.edu/~sco590/appendix_k_standalone.pdf`, accessed on September 21th, 2020.

4.4 Foreground-background separation

Q10.1: As Figure 9, The algorithm first smooths the image and utilizes the k-means clustering algorithm to extract the cluster of pixels belonging to the foreground. Using the foreground cluster the foreground can be separated from the background. With the standard parameter settings the algorithm performs good an the images *Polar*, *Robin-1* and *Robin-2*. The images *Cows*, *Kobi* and *SciencePark* provide some issues. The *Kobi* image seems to perform worse because the shade of the dog on the floor and the *Cows* image seems to perform worse because the head of the calf is significantly darker in comparison to the rest of the cows. The *SciencePark* image does provide one of the segments as foreground and in theory the small protrusion from the wall are closer to the camera but as we think this image should be seen as one plane we considered this a sub optimal outcome.

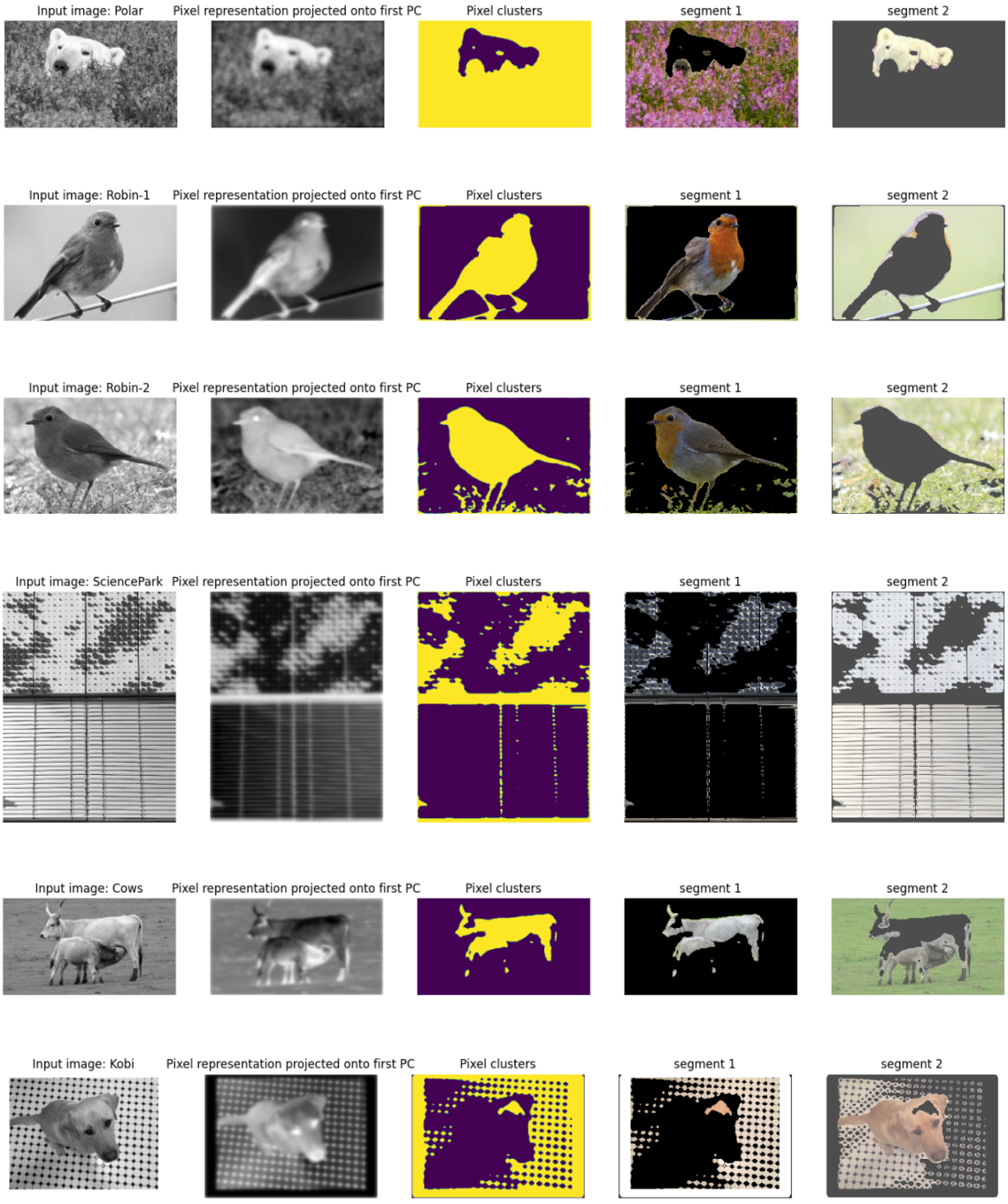


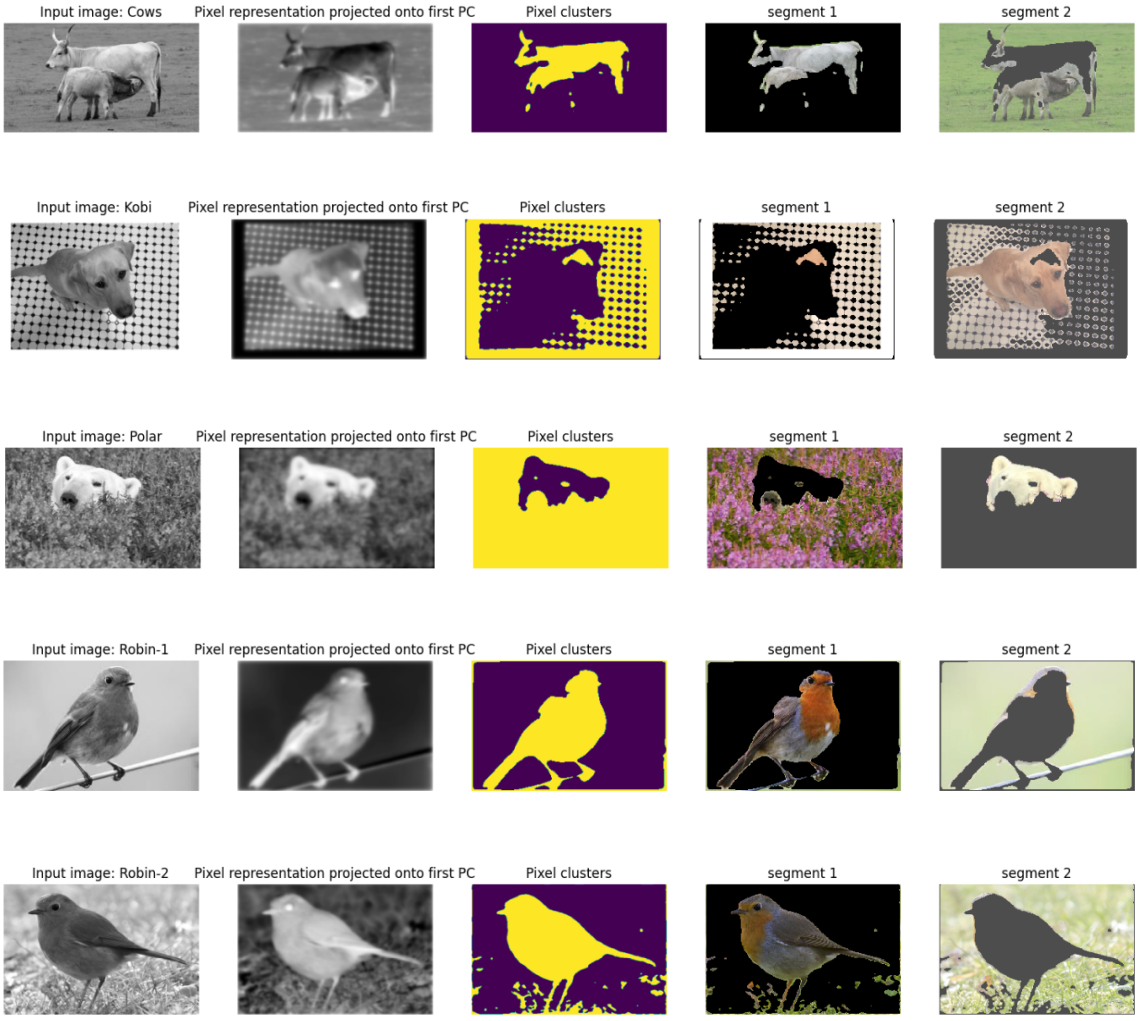
Figure 9: Segment results with default parameter

Q10.2: As shown in the figure above, all segments work fine expect cows and Kobi. After doing experiments with different σ, λ and θ, we find that λ and θ do not have huge influence on segmentation result. However, the value of sigma help improve Kobi results. We change the sigma form

[1,2] to [0.2,0.4], then the the result segment Kobi improve as shown in Figure 10, though spots on the floor is still not fully recognizable.

Figure 10: Segment results with improved parameter

Q10.3: After turning off the *smoothingFlag* , two phenomena happen as shown in Figure 11. The first is some pixels which were in same cluster change to the other cluster. The second is some edges and corners or noises emerge. The reason is in smoothing, the data points of a signal are modified so individual points (presumably because of noise) are changed, and points that are different from the adjacent points are also changed leading to a smoother signal. This produces pixels with intermediate values between cluster means. When smooth is off, these phenomena problems happen.



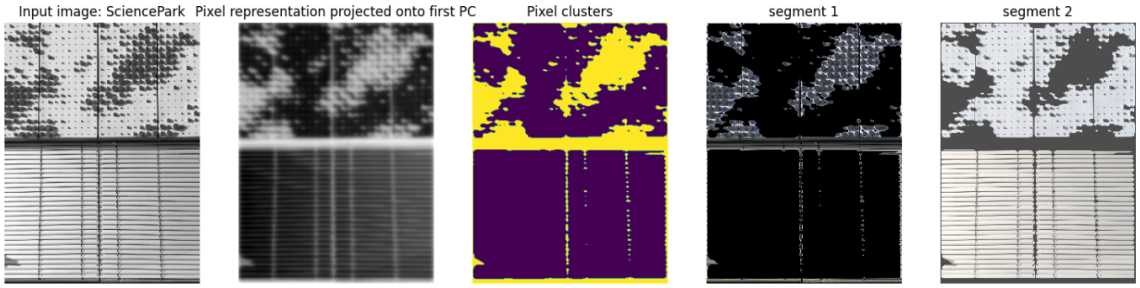


Figure 11: Segment results without smoothing

5 Conclusion

In our section on convolutions it is shown that while convolutions and correlations are essentially similar, the convolutions are more convenient and efficient to use. We have applied the theory of convolutions by creating and experimenting with Gaussian filters, of which primarily the second derivative of a Gaussian filter as it has applications for edge detection. We also further investigated Gabor filters as they approach edge detection in a different manner. We look into how Peak Signal to Noise Ratio is used to measure the effects filters have on images. Finally we implemented these various techniques to create edge detection and foreground-background separation.

References

Derpanis, K. G. (2006). Outline of the relationship between the difference-of-gaussian and laplacian-of-gaussian. *NYU Dept. of CS & Eng*, pages 1–3.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110.

Marr, D. and Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217.

Winnemöller, H., Kyprianidis, J. E., and Olsen, S. C. (2012). Xdog: an extended difference-of-gaussians compendium including advanced image stylization. *Computers & Graphics*, 36(6):740–753.